МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

# ОТЧЕТ
по лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
Тема : Рекурсии
Вариант №26

Студент гр. 8304 Завражин Д. Г.
Преподаватель Фирсов М. А.

Санкт-Петербург
2019

## 1. Цель работы

Функция Ф преобразования последовательности α ={a1, a2, …, an} натуральных чисел в последовательность целых чисел определена следующим образом:

• Ф({})={0};

• Ф({a})={-a};

Если ||α||>1 (т.е. в последовательности больше одного натурального числа), то α = {a} U β = {a} U {$b_1$, $b_2$, …, $b_m$} (U – знак конкатенации). Тогда:

• Ф(α) = {666}, если ∄ i: a ⋮ bi (a ⋮ b означает, что a делится нацело на b);

• Ф(α) = Ф({b1, b2, …, bi-1}) U {999, (a+bi)} U Ф({bi+1, bi+2, …, bm}), где a ⋮ bi и ∄j: (j<i, a ⋮ bj).

Примеры:

Ф({2, 1})=Ф({}) U {999, (2+1)} U Ф({}) ={0, 999, 3, 0}

Ф({5, 2, 3})={666}

Ф({1, 2})={666}

Ф({12, 5 , 5, 6, 8}) = Ф({5, 5}) U {999, (12+6)} U Ф({8}) = Ф({}) U {999, (5+5)} U Ф({}) U {999, 18} U {-8} = {0} U {999, 10} U {0} U {999, 18, -8} = {0, 999, 10, 0,
999, 18, -8}

Реализовать функцию Ф рекурсивно.

## 2. Входные данные

Входными данные являются строки, находящиеся в файле tests.txt.

## 3. Описание программы

Входными данные являются строки, находящиеся в файле tests.txt.

### 3.1. Структуры данных

Для обеспечения представления последовательности была создана структура данных Sequence на основе динамического массива. Исходный код данной структуры данных приведён ниже.

```
template<class T>
struct Sequence
{
	size_t size = 0;
	size_t capacity = initial_sequence_capacity;

	T *array = new T[initial_sequence_capacity];

	~Sequence()
	{
		delete [] array;
	}

	// overload assignment operator
```

```
	void operator=(const Sequence<T> &seq)
	{
		delete [] array;
		this->size = seq.size;
		this->capacity = seq.capacity;
		this->array = new T[this->capacity];
		for(size_t i = 0; i < this->size; i++)
			this->array[i] = seq.array[i];
	}

	// append the given element(s)
	Sequence<T> &append(T elem)
	{
		if(this->size >= this->capacity)
		{
			this->capacity *= 2;
			T *tmp_array = (T *) realloc(this->array, this->capacity *
sizeof(T));
			if(tmp_array == nullptr)
				throw std::bad_alloc();
			this->array = tmp_array;
		}
		this->array[this->size] = elem;
		++this->size;
		return *this;
	}

	Sequence<T> &append(const Sequence<T> &elems)
	{
		for(size_t i = 0; i < elems.size; ++i)
			this->append(elems.array[i]);
		return *this;
	}

	template<typename... Ts>
	Sequence<T> &append(T elem, Ts...elems)
	{
		this->append(elem);
		this->append(elems...);
		return *this;
	}

	template<typename... Ts>
	Sequence<T> &append(const Sequence<T> &elems1, Ts...elems2)
	{
```

```
        this->append(elems1);
        this->append(elems2...);
        return *this;
}

//overload the costructor to construct a sequence with provided elements
Sequence(){}

Sequence(T elem)
{
        this->append(elem);
}

Sequence(const Sequence<T> &elems)
{
        this->append(elems);
}

template<typename... Ts>
Sequence(T elem, Ts...elems)
{
        this->append(elem, elems...);
}

template<typename... Ts>
Sequence(const Sequence<T> &elems1, Ts...elems2)
{
        Sequence();
        this->append(elems1, elems2...);
}

// some convenient member functions
Sequence<T> left_third(size_t pos)
{
        Sequence<T> result;
        // 1 here assures that the sequence-initial element
        // is not considered to be a part of the left segment
        for(size_t i = 1; i < pos; ++i)
                result.append(this->array[i]);
        return result;
}

Sequence<T> right_third(size_t pos)
{
        Sequence<T> result;
```

```
            for(size_t i = pos + 1; i < this->size; ++i)
                result.append(this->array[i]);
            return result;
        }

        // print the visual sequence representation
        void print()
        {
            cout << '{';
            if(this->size > 0)
                cout << this->array[0];
            for(size_t i = 1; i < this->size; i++)
                cout << ", " << this->array[i];
            cout << '}';
        }
};
```

### 3.2. Зависимости и объявление констант и функций

```
#include <cstdlib>      // for memory reallocation
#include <iostream>   // for basic input-output functionality
#include <fstream>     // for file input-output functionality
#include <string>       // for storage of lines read from a file
#include <sstream>     // for conversion from a string to long long unsigned

constexpr size_t initial_sequence_capacity = 10;
constexpr bool print_recursion_information = true;

Sequence<long long signed> Phi(Sequence<long long unsigned> &seq);
void readFromFile(const std::string &file_path);
int main();
```

### 3.3. Функция main

Функция *main* запрашивает у пользователя путём к файлу tests.txt, содержащему входные данные для вызовов функции Phi и вызывает функцию *readFromFile* с полученным от пользователя путём.
Исходный код функции *main* приведён ниже.

```
int main()
{
    std::string file_path = "";
    cout << "Enter path to a file containing test cases." << endl;
    cin >> file_path;
    readFromFile("tests.txt");
    return 0;
```

```
}
```

### 3.4. Функция readFromFile

Функция *readFromFile* обеспечивает чтение из файла с построчным извлечением из него текстовых представлений последовательностей, интерпретирует их и вычисляет от них функцию *Phi* с последующим выводом результата в консоль.
Исходный код функции *readFromFile* приведён ниже.

```
void readFromFile(const std::string &file_path)
{
	// used to output debug information
	size_t line_counter = 0;

	std::ifstream file(file_path);

	// each line in a file is interpreted to contain a sequence
	if (file.is_open())
	{
		for(std::string line; getline(file, line);)
		{
			cout << endl;
			cout << "Line " << line_counter << endl;
			++line_counter;
			Sequence<long long unsigned> seq;
			long long unsigned number;

			auto current = line.begin();
			auto end = line.end();
			std::string number_representation = "";

			if(current == end)
			{
				cout << "Error: line " << line_counter << " happens to be empty." << endl;
				continue;
			}
			if(*current != '{')
			{
				cout << "Error: line " << line_counter << " does not start with '{'." << endl;
				continue;
			}
			++current;
```

```
                bool curly_bracket_encountered = false;
                while(current != end)
                {
                        if(std::string("0123456789").find(*current) !=
std::string::npos)
                        {
                                number_representation += *current;
                        }
                        else if(*current == ',')
                        {
                                if(number_representation.length() > 0)
                                {
                                        std::stringstream(number_representation) >>
number;
                                        if(number == 0)
                                        {
                                                cout << "Error: line " << line_counter <<
" contains the number 0, which is not a natural number." << endl;
                                                goto new_line;
                                        }
                                        seq.append(number);
                                        number_representation = "";
                                }
                        }
                        else if(*current == '}')
                        {
                                if(number_representation.length() > 0)
                                {
                                        std::stringstream(number_representation) >>
number;
                                        if(number == 0)
                                        {
                                                cout << "Error: line " << line_counter <<
" contains the number 0, which is not a natural number." << endl;
                                                goto new_line;
                                        }
                                        seq.append(number);
                                        number_representation = "";
                                }
                                curly_bracket_encountered = true;
                                break;
                        }
                        else if(*current == ' ' || *current == '\t')
                        {
                                if(number_representation.length() > 0)
```

```
                                {
                                        std::stringstream(number_representation) >>
number;
                                        if(number == 0)
                                        {
                                                cout << "Error: line " << line_counter <<
" contains the number 0, which is not a natural number." << endl;
                                                goto new_line;
                                        }
                                        seq.append(number);
                                        number_representation = "";
                                }
                        }
                        else
                        {
                                cout << "Warning: line " << line_counter << "
contains an unsupported character '" << *current << "' which will be ignored." <<
endl;
                        }
                        ++current;
                }
                if(!curly_bracket_encountered)
                {
                        cout << "Error: line " << line_counter << " happens to lack
'}'." << endl;
                        continue;
                }
                cout << "The following sequence was acquired:" << endl;
                seq.print();
                cout << endl;
                cout << "Result:" << endl;
                Phi(seq).print();
                cout << endl;
                new_line:
                continue;
        }
    }
    else
    {
        cout << "Unfortunately, file could not be opened." << endl;
    }
}
```

### 3.4. Функция Phi

Функция *Phi* производит преобразование последовательности натуральных чисел в последовательность целых согласно поставленной задаче.
Исходный код функции *Phi* приведён ниже.

```
Sequence<long long signed> Phi(Sequence<long long unsigned> &seq)
{
	// the depth will be tracked using a static variable
	static size_t depth = 0;
	++depth;
	if(print_recursion_information)
	{
		cout << std::string(depth - 1, ' ') << "|-----------" << endl;
		cout << std::string(depth - 1, ' ') << "| Phi was called with seq = ";
		seq.print();
		cout << endl;
		cout << std::string(depth - 1, ' ') << "| depth: " << depth << endl;
		cout << std::string(depth - 1, ' ') << "| cardinality : " << seq.size << endl;
	}

	Sequence<long long signed> result;
	// in the case of an empty sequence, {0} shall be returned
	if(seq.size == 0)
	{
		if(print_recursion_information)
			cout << std::string(depth - 1, ' ') << "| empty sequence case" <<
endl;
		result.append(0);
	}
	// in the case of a sequence containing a single value,
	// a sequence containing only the one opposite to it shall be returned
	else if(seq.size == 1)
	{
		if(print_recursion_information)
			cout << std::string(depth - 1, ' ') << "| sole element case" << endl;
		result.append(-(long long)seq.array[0]);
	}
	else
	{
		// the program shall check whether the first number in the given
sequence
		// is divisible by at least one another
		bool div = false;
		size_t pos = 1;
		for(size_t i = 1; i < seq.size && !div; ++i)
```

```
                if(seq.array[0] % seq.array[i] != 0)
                        ++pos;
                else
                        div = true;
        if(div)
        {
                if(print_recursion_information)
                {
                        cout << std::string(depth - 1, ' ') << "| tripartite case" <<
endl;
                        cout << std::string(depth - 1, ' ') << "| pos  : " << pos <<
endl;
                }
                auto left_third = seq.left_third(pos);
                auto right_third = seq.right_third(pos);
                result.append(Phi(left_third));
                result.append(999, seq.array[0] + seq.array[pos]);
                result.append(Phi(right_third));
        }
        else
        {
                if(print_recursion_information)
                        cout << std::string(depth - 1, ' ') << "| 666 case" << endl;
                result.append(666);
        }
    }
    if(print_recursion_information)
    {
        cout << std::string(depth - 1, ' ') << "|-----------" << endl;
        cout << std::string(depth - 1, ' ') << "| depth: " << depth << endl;
        cout << std::string(depth - 1, ' ') << "| Phi exited with value ";
        result.print();
        cout << endl;
    }
    --depth;
    if(depth == 0 && print_recursion_information)
        cout << "|-----------" << endl;
    return result;
}
```

### 3.4. Тестирование программы

Тесты, содержащиеся в файле tests.txt, и выжные с точки зрения оценки работы программы фрагменты её вывода приведены в таблице 1.

Таблица 1. Тесты, применённые при тестирование программы.

| Тест | Вывод программы |
|---|---|
| | Error: line 1 happens to be empty. |
| Test | Error: line 2 does not start with '{'. |
| { | Error: line 3 happens to lack '}'. |
| {45 | Error: line 4 happens to lack '}'. |
| {45, 34 | Error: line 5 happens to lack '}'. |
| {} | Result: {0} |
| {5} | Result: {-5} |
| {-5} | Warning: line 8 contains an unsupported character '-' which will be ignored.<br>The following sequence was acquired: {5}<br>Result: {-5} |
| {-5.7} | Warning: line 9 contains an unsupported character '-' which will be ignored.<br>Warning: line 9 contains an unsupported character '.' which will be ignored.<br>The following sequence was acquired: {57}<br>Result: {-57} |
| {0} | Error: line 10 contains the number 0, which is not a natural number. |
| {0, 0} | Error: line 11 contains the number 0, which is not a natural number. |
| {2, 1} | Result: {0, 999, 3, 0} |
| {5, 2, 3} | Result: {666} |
| {1, 2} | Result: {666} |
| {12, 5, 5, 6, 8} | Result: {0, 999, 10, 0, 999, 18, -8} |
| {12, 5, 5, 6, 8, 8} | Result: {0, 999, 10, 0, 999, 18, 0, 999, 16, 0} |
| {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1} | Result: {0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0, 999, 2, 0} |

На всех приведённых выше входных данных программа выдаёт ожидаемый результат; таким образом можно сделать вывод, что данная программа корректно работает во всех охватываемых составленными тестами случаях.

**4. Вывод**

В результате выполнания лабораторной работы была реализована программа, отвечающая всем поставленным условиям и проходящая рассмотренное выше составленное в процессе выполнения работы тестирование. Помимо этого, были на практическом примере отточены навыки проектирования, написания и тестирования рекурсивных алгоритмов, владения языком C++, прямой работы с динамическими массивами.